オペレーティング・システム 42 － 1
（１９８９．２．２４）

ＯＳ／omicron 第２版の実現と評価

鈴木茂夫，田中泰夫，岡野裕之，堀素史，横関隆，並木美太郎，高橋延匡

東京農工大学　工学部　数理情報工学科

本報告では，研究用計算機システムＯＳ／ｏ第２版におけるタスク管理の設計と実現，および，その性能評価について述べる．我々は，日本語情報処理を目的としてフル２バイトコード系を持ち，研究者が自由に手を加えることのできる透明性を持ったＯＳを目標としてＯＳ／ｏ第２版の開発を行なった．本研究では，上記の目標に対して次に述べるものを実現した．

（１）並列処理を目的として，マルチタスク機能を実現した．タスクのスケジューリングについては，その操作を全面的にユーザに開放する．また，密な関係にあるタスク間での，柔軟でかつ高速な情報交換を可能とするため，一部の実行環境を共有したタスクの形態(タスクフォース)を実現した．

（２）ＯＳの機能を動的に拡張できる機構(ユーザ拡張部)を実現した．また，ユーザ固有のデバイスハンドラの登録を可能とした．この機構を利用して日本語入力機能の実現を行なった．

本研究により，新しいデバイスを用いたマンマシンインタフェースの研究，そして並列処理記述言語などの研究の基盤となる環境を提供することができた．

Design and Implementation of Task Management on OS/omicron V2.2

Shigeo SUZUKI, Yasuo TANAKA, Hiroyuki OKANO, Motofumi HORI,
Takashi YOKOZEKI, Mitarou NAMIKI and Nobumasa TAKAHASHI

Tokyo University of Agriculture and Technology
2-24-16, Naka-machi, Koganei, Tokyo, 184, Japan

In this paper, We present design, implementation and performance evaluation of the task manager of OS/omicron Version 2.2. OS/omicron is based on the Japanese character code. Its transparency and extensibility provide users with freedom to enhance OS/omicron for their own use.

Main results of this research are:

1. The task manager provides multi-tasking for parallel processing, flexible task-scheduling and "task force" that shares a part of executing environment for tightly coupled inter-task communication.
2. Operating system facilities can be dynamically enhanced, so that handlers of external devices can be registered. We implemented Japanese input facilities by this mechanism.

In this research, a flexible research environment has been realized in which man-machine interface with new devices and parallel programming can be studied.

１．はじめに

我々は，日本語情報処理を目的とした研究用計算機システムＯＳ／omicron の研究・開発を行なっている．本システムは，研究室で行なわれているオンライン手書き文字認識による日本文入力の研究，レーザビームプリンタを用いた日本語文書出力システムの研究，日本文処理の研究，そして並列処理記述言語の研究の基盤となる環境を提供することが主目的である．したがって，研究対象に合うように，ＯＳに対して機能の追加，削除といった改造を，研究者が自由に行なえるように設計した．また，日本語処理を可能とするため，文字コードとして JIS X 0208 のフル２バイトコード系を採用した[1]．これにより，日本語の文字を，何ら特殊扱いすることなく，通常の英数字と同様の文字として，自然な形で処理できるようになった．そして，言語処理系の識別子にいたるすべてのリソース名に対して，日本語の使用を可能とした．

また，ＯＳ／ｏでは，オンライン手書き文字認識といったリアルタイムアプリケーションの時間的制御を容易にするため，実記憶方式を採用している．このため，CPU としてアドレス空間の広い MC68000 シリーズを選択した．そして，マルチプロセッサに対応するため，オブジェクトコードはリロケータブルでかつリエントラントなものを標準とした．このオブジェクトコードは，ＯＳ／ｏのシステム記述言語の処理系として開発した日本語言語ＣコンパイラＣＡＴ（C compiler developed at Tokyo University of Agriculture and Technology）により生成される[2]．

本報告では，上記の特徴を持つＯＳ／ｏ第２版におけるタスク管理の実現とその性能評価について述べる．

２．ＯＳ／ｏ第２版のシステム構成

ＯＳ／ｏ第２版では，ユーザが記述する通常のアプリケーションプログラムから構成される層のことを，ユーザアプリケーション層と呼ぶことにする．このユーザアプリケーション層とＯＳの核の間に一つの層を設け，ＯＳの機能を動的に拡張する機構を実現した．したがって，ユーザアプリケーション層からは，より拡張されたＯＳが見えることになる．この拡張機能から構成される層のことをユーザ拡張部（システムユーザ向けＯＳ設定部）と呼ぶ（図１参照）．

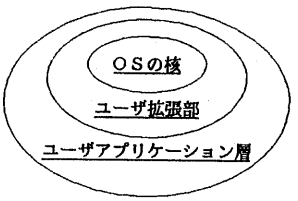


図１　ＯＳ／ｏ第２版のシステム構成

ＯＳの核はタスク管理部，ファイルシステム部，メモリ管理部，そして日本語辞書ハンドラ部から構成される．ユーザ拡張部の登録機構は，ＯＳの核のタスク管理部により管理される．タスク管理部の中でタスクのスケジューリング操作を行なう部分のことを，特にディスパッチャと呼ぶことにする．

３．タスク管理の特徴

次に，ＯＳ／ｏ第２版におけるタスク管理部の特徴について述べる．

３．１　タスク管理開放型マルチタスク方式

並列処理の実現と，プログラミング環境におけるマルチタスクの有用性を考慮して，ＯＳ／ｏ第２版ではマルチタスク機能を実現した．このマルチタスク方式は，独占的使用での性能を最大限に発揮できることを目標とする．したがって，タスクのスケジューリングに関しては，その操作を全面的にユーザに開放する．具体的には，タスクの優先順位を変更する機能，タスク間同期命令，そしてハードウェアタイマによる時間的制御機能などをユーザプログラムに提供する．また，タスク管理部のスケジューリング機能そのものをソースコードレベルで改造することを許す．

３．２　すべての処理プログラムをタスクとして管理

ＯＳ／ｏ第２版では，すべての処理プログラムをタスクという単位で統一的に管理する．ユーザプログラムは，ユーザアプリケーション層においてタスクとして生成され，実行される．これをユーザタスクと呼ぶ．そして，このユーザタスクから呼び出されるＯＳ自身も，タスクとしての実行環境を持ち，スケジューリングの対象となって処理を進める．また，外部割込みをタスク間同期命令に抽象化することにより，その処理プログラムもタスクとして実行される．これらタスクとして生成された処理プログラムは，タスク管理より提供される統一したインタフェース（タスク間同期，通信機能）によって互いに作用し合いながら処理を進める．

３．３　ＯＳの動的な機能拡張

ＯＳ／ｏ第２版では，ＯＳの機能を拡張する機構をユーザ拡張部という形で実現した．これにより，ＯＳ／ｏを核としたシステムの構築の際，そのシステムにとって標準的な機能，また保護の強い機能をＯＳの拡張機能として自由に登録することができる．また，新しいデバイスに対するハンドラを登録することも可能である．これらの拡張機能は，動的に登録，削除することができる．したがって，新しいデバイス，例えば透明タブレットを用いたマンマシンインタフェースの研究などを行なう際に，そのデバイスを扱う機能を変化させ，実験的に評価するといった応用も可能である．

3．4　同期基本命令としてセマフォを採用

ＯＳ／ｏ第２版では，タスク間同期基本命令としてセマフォを採用している．これは，セマフォがもっともプリミティブな同期命令であり，他のタスク間同期機能はこのセマフォを用いて実現できるという理由からである．タスクとして実現されるＯＳ自身の同期処理は，すべてこのセマフォを用いて記述されている．

ユーザタスク間で利用する同期，通信機能として，より抽象度の高い機能であるメッセージ通信機能を実現した．しかし，この通信方式は，抽象化された機能であるがために，プログラムの生産性，保守性がよい反面，自由度が低いといった欠点を持つ．そこで，ＯＳ／ｏ第２版では，ユーザタスクに対しても，セマフォの直接の使用を許すことにした．

3．5　タスクフォースの実現

密な関係にあるタスク群において，高速で，かつ柔軟な情報交換を可能とするため，データ領域などの一部の実行環境を共有したタスクの形態を実現した．これをタスクフォースと呼ぶ．タスクフォースを利用することにより，種々の通信方式のシミュレーション的な実験，そして並列処理記述言語の研究を行なうといった応用も可能となる．

4．ユーザタスクの生成と実行

本章では，ユーザアプリケーション層において，ユーザプログラムがどのようにタスクとして生成，実行されるかを中心に述べる．

4．1　タスクとタスクフォース

4.1.1　タスクの実行環境

タスクは次に示す実行環境を持つ．

（1）主記憶上の４領域

タスクは主記憶上に手続き領域，静的データ領域，ヒープ領域，そしてスタック領域の４領域を持つ．これらの実行環境は言語Ｃ処理系ＣＡＴにより生成され，リロケータビリティ，リエンタラビリティを保証している．それぞれの領域へのアクセスはベースレジスタを介して行なう．ベースレジスタとして MC68000 のアドレスレジスタ(A3～A6)を利用している．また，スタックポインタ，スタックベースとして A1，A2 レジスタを使用している（図２参照）．

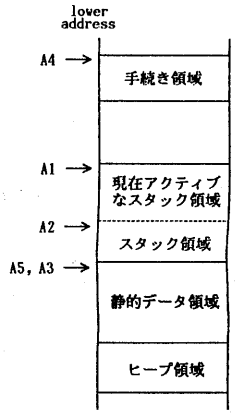


図２　タスクの実行環境

（2）プロセッサのレジスタ

MC68000 プロセッサのレジスタである PC，SR レジスタ，アドレスレジスタ，データレジスタを実行環境として持つ．

（3）セマフォ，メッセージ

ＯＳ／ｏ第２版ではタスク間同期，排他制御の手段としてセマフォを採用した．そして，その直接の使用をユーザプログラムに開放した．また，タスク間同期，通信手段としてメッセージ通信を実現した．これらセマフォ，メッセージはタスクの実行環境となる．

（4）カレントディレクトリ，ファイル

ＯＳ／ｏ第２版では，ファイルシステムにより階層構造のディレクトリを実現した．タスクが現在使用しているディレクトリの位置，および使用しているファイルを実行環境として持つ．

4.1.2　タスクフォースで共有する実行環境

タスクは，基本的にはそれぞれ個々の実行環境を持ち，それぞれ独立して実行を進める．しかし，目的とする仕事によっては，複数のタスクが何らかの影響を互いに与え合いながら処理を進める形態が考えられる．こうしたタスク間のデータ交換および同期に関する機能は，当然ＯＳにより提供される．しかし，ＯＳの機能の呼出しは，コンテキストスイッチングを伴うため，その回数が多くなればなるほど処理全体の効率に大きな影響を与える．そこで，ＯＳ／ｏでは，一つの仕事を複数のタスクが共同して行なうような形態を，効率良く実現することを目的として，一部の実行環境を共有したタスクフォースと呼ばれる形態を採り入れた．

タスクフォースでは，主記憶上の４領域のうち手続き，静的データ，ヒープの３領域，セマフォ，メッセージ，そしてファイルを共有する．したがって，静的データ領域を共有データ領域として，またもっともプリミティブな同期，排他制御の手段であるセマフォを利用することにより，柔軟でかつ高速なデータ交換が可能となる．

タスクフォースで共有する実行環境は TFCB (Task Force Control Block) により，個々のタスクで持つ実行環境は TCB (Task Control Block) によりそれぞれ管理される．

4.1.3 タスクの生成

すべてのユーザタスクは，いずれかのタスクフォースに属している．したがって，自タスクフォースの外にタスクを生成する場合には，まず，新しいタスクフォースを生成する必要がある．この機能はＯＳにより提供される．タスクフォース生成機能を呼び出すと，タスクフォースで共有する実行環境が生成され，同時にタスクフォース内でルートにあたるタスクも生成される．このタスクをタスクフォースリーダと呼ぶ．タスクフォースリーダがタスクフォース内にタスクを生成した場合には，それらをタスクフォースメンバ（または単にタスク）と呼ぶことにする．

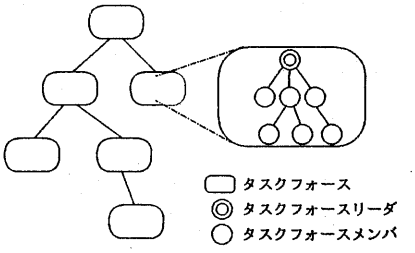


図３　タスク，タスクフォースの生成関係

タスク，そしてタスクフォースの生成関係は，図３に示すような木構造の親子関係になる．タスクフォースの生成は __create_tf（表１参照）を呼び出すことにより行なわれる．引数は，ロードモジュール名，およびタスクフォース生成に必要な情報パケットである．情報パケットの内容は，タスクフォースで使用するスタック，ヒープ領域などの大きさ，および子タスクフォースに引き継ぐ実行環境の指定である．__create_tf を呼び出すと，引数の情報にしたがって，タスクフォースで共有する実行環境，そしてタスクフォースリーダが生成される．戻り値としてタスクフォース識別子が返される．

タスクフォースメンバの生成は，__create_task により行なわれる．引数は，タスクのエントリポイント，タスクが使用するスタックの大きさ，そしてタスクの種類である．タスクのエントリポイントとは，タスクフォースで共有する手続き領域内でのタスクの実行開始位置を指す値であり，言語Ｃの関数ポインタを用いて指定を行なう．タスクの種類には，通常のタスクとエラー例外処理タスクがある．タスクフォース内で，エラー例外に対する処理を行なうタスクフォースメンバをエラー例外処理タスクと呼ぶ．戻り値としてタスク識別子が返される．

4.1.4 タスクの状態

生成されたタスクは，基本的には図４に示すように Ready 状態，Running 状態，Suspended 状態の３つの状態を遷移して実行を進める．Ready 状態のタスクは Ready リストにリンクされて管理される．Ready リストはタスクの持つ優先順位の高い順にソートされている．優先順位のもっとも高い値は，外部割込み処理タスクのために予約され，もっとも低い値はアイドリングタスクのために予約されている．

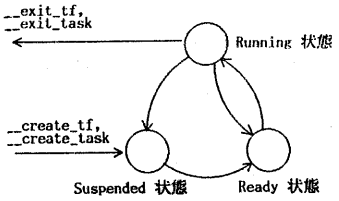


図４　タスクの状態遷移

ディスパッチャは，現在実行可能な状態になっているタスクのうち，もっとも優先順位の高いものにプロセッサを割り当てる．つまり，Ready リストにリンクされているタスクのうち，先頭にリンクされているタスクが Running 状態となる．Running 状態のタスクは，Ready リストの先頭にリンクされたままで管理される．これは，ディスパッチャの実行効率を考慮した結果の仕様である．

Suspended 状態とは，事象の発生を待っているタスクの状態を指す．ＯＳ／ｏ第２版では，タスクの同期手段としてセマフォを採用している．したがって，ＯＳ／ｏ第２版における事象とは，Ｖ命令の実行，または外部割込みの発生である．Suspended 状態のタスクは，セマフォの Suspended リストまたは外部割込み待ちリストにリンクされて管理される．

4.1.5 親子タスク間の同期機能

タスクは生成しただけでなく，そのタスクを実行させたり，またその結果の値を受け取るといった処理が必要になる．ＯＳ／ｏ第２版では，このような親子タスク間の同期，データ交換の機能を，ＰＶ命令などのプリミティブな同期手段とは別に，より抽象度の高い形でユーザタスクに提供する．また，親子タスクフォース間でも同様の機能を提供する．

生成した子タスクは，__go_task により実行される．引数として生成した子タスクのタスク識別子を指定する．自タスクの実行を終了し，処理結果の値を親タスクに返す機能が __exit_task であり，引数として親タスクに返す値を指定する．また，自タスクの実行をいったん停止して，途中結果の値を親タスクに返す機能として __stop_task がある．そして，子タスクの実行が終了するのを待ち，その結果の値を受け取る機能が __wait_task である．__wait_task は，指定した子タスク，または任意の子タスクの終了を待つことができる．これらが基本となる親子タスク間の同期機能であるが，もう一つ特殊な機能として，子タスクの実行を強制的に終了させる __kill_task がある．親子タスク間の同期機能により，ユーザは Ready 状態，Running 状態，Suspended 状態の３状態より抽象化された状態遷移で，タスクの実行をとらえることができる．

4．2　タスク間通信機能

ＯＳ／ｏ第２版におけるタスク間通信機能について述べる．

4.2.1　タスクフォース内でのタスク間通信

タスクフォースでは静的データ領域を共有している．したがって，タスクフォース内の各タスクは，この領域を利用して自由にデータ交換を行なうことができる．そして，この共有データ領域をアクセスする際の排他制御手段としてセマフォを使用することができる．

セマフォは，__create_sem により生成される．戻り値としてセマフォ識別子が返される．ＰＶ命令を実行する機能である __P_op，__V_op およびセマフォを削除する機能である __delete_sem は，引数としてこのセマフォ識別子を指定する．セマフォは，生成されたタスクフォース内だけでその使用が許される．つまり，他のタスクフォースで生成されたセマフォに対してＰＶ命令，また削除を実行することはできない．セマフォは各タスクフォースごとに AST (Active Semaphore Table) により管理され，タスクフォース内で一意のセマフォ識別子を持つ．

4.2.2　メッセージ通信機能

タスクフォース内でのタスク間と異なり，別のタスクフォースに属するタスク間では共有のデータ領域を持たない．これらのタスク間での通信機能として，ＯＳ／ｏ第２版では，同期式のメッセージ通信機能を実現した．

メッセージ通信経路は，__create_mssg により生成される．引数に通信相手，通信モード (RECEIVE/SEND) を指定する．このとき，通信相手として指定する値は，通信相手のタスクフォース識別子である．したがって，このメッセージ通信はタスクフォース間での通信手段としてとらえることができる（図５参照）．

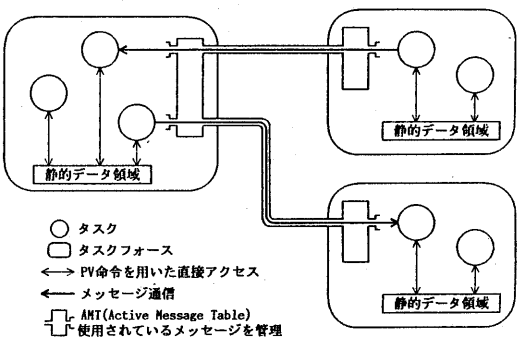


図５　メッセージ通信

通信を行なう２つのタスクフォースが，お互いに対応したモードで __create_mssg を呼び出した場合に，メッセージの経路が結ばれる．通信相手を指定する引数の値を -1 とすることにより，通信相手に任意のタスクフォースを指定することができる．この場合，最も速く自タスクフォースに対してメッセージ通信経路を生成したタスクフォースと通信経路が結ばれる．

__create_mssg によりメッセージ通信経路が結ばれると，戻り値としてメッセージ識別子が返される．このメッセージ識別子をもってメッセージの送受信を行なう．この機能が __send_mssg，__receive_mssg である．メッセージの送受信は同期式である．もし，送受信処理を非同期に行ないたい場合には，送受信処理を行なうタスクフォースメンバを生成し，それを非同期に実行されることにより実現できる．

表1　ユーザタスクが使用できるＳＶＣライブラリ関数

関数名	機　　能
__create_tf	子タスクフォースを生成する
__go_tf	子タスクフォースを実行させる
__stop_tf	自タスクフォースの実行を停止する
__exit_tf	自タスクフォースの実行を終了する
__wait_tf	子タスクフォースの停止，終了を待つ
__kill_tf	子タスクフォースを削除する
__create_task	子タスクを生成する
__go_task	子タスクを実行させる
__stop_task	自タスクの実行を停止する
__exit_task	自タスクの実行を終了する
__wait_task	子タスクの停止，終了を待つ
__kill_task	子タスクを削除する
__create_sem	セマフォを生成する
__delete_sem	セマフォを削除する
__P_op	Ｐ命令を実行する
__V_op	Ｖ命令を実行する
__create_mssg	メッセージ通信経路を生成する
__delete_mssg	メッセージ通信経路を削除する
__receive_mssg	メッセージを受信する
__send_mssg	メッセージを送信する
__set_ue_task	ユーザ拡張部の登録を行なう
__reset_ue_task	ユーザ拡張部の削除を行なう

関数名	機　　能
__self_task_id	自タスク識別子を得る
__self_tf_id	自タスクフォース識別子を得る
__root_tf_id	ルートタスクフォース識別子を得る
__leader_task_id	リーダタスク識別子を得る
__tf_family	タスクフォースの親子関係を得る
__task_family	タスクの親子関係を得る
__get_tf_size_inf	各領域の大きさ情報を得る
__get_tf_base_inf	各領域の基底アドレス情報を得る
__get_error_inf	エラー例外情報を得る
__get_tf_name	タスクフォース名を得る
__set_tf_name	タスクフォース名を設定する
__tf_name_to_id	名前からタスクフォース識別子を得る
__get_module_name	モジュール名を得る
__task_status	タスクの状態を得る
__get_stack_inf	スタック領域に関する情報を得る
__set_stack_pointer	スタックポインタを設定する
__get_rgstr	レジスタ情報を得る
__set_rgstr	レジスタ値を設定する
__get_priority	優先順位を得る
__set_priority	優先順位を設定する
__trace_on	トレース状態に設定する
__trace_off	トレース状態を解除する

4.2.3 親子タスクフォース間でのポインタ通信

通常のタスクフォース間では，データを共有することができない．しかし，親子関係にあるタスクフォース間で，しかも親タスクフォースがそれを許可した場合にのみ，データの共有を許すことにした．これは，親子関係にあるタスクフォース間では，通常のタスクフォース間と異なり，密接なデータ交換を必要とする場合が予想されるためである．共有するデータ領域は，親タスクフォースの静的データ領域，または子タスクフォースの静的データ領域である．また，子タスクフォースをデバッグ対象とするデバッガを実現可能にするため，子タスクフォースの手続き領域，スタック領域を親タスクフォースのデータ領域としてアクセス可能とする．これらデータの共有方式の指定は，子タスクフォース生成時に親タスクフォースが行なう．また，このとき同時に親子タスクフォースで共有するセマフォの指定を行なう．

子タスクフォースを生成する際にデータの共有を指定した場合には，親タスクフォースと子タスクフォースのポインタの基底アドレスが共通になる．したがって，共有データ領域を指すポインタ値を，メッセージ通信，または親子タスクフォース間同期機能などを利用して，通信相手のタスクフォースに送ることにより，共有データとセマフォを用いた高速で柔軟なデータ交換が可能となる．こうしたポインタ値を介しての通信方式をポインタ通信と呼ぶ．

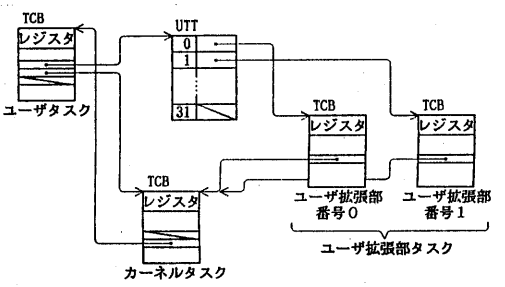
5．ＯＳの核およびユーザ拡張部の実現

前章までに，ユーザプログラムがどのようにタスクとして生成，実行されるかについて述べてきた．本章では，ＯＳの核の実行をどのように扱うかについて，そしてユーザ拡張部の実現について述べる．

5．1　ＯＳの核の実現

5.1.1　ＯＳのタスク化

ＯＳ／ｏ第２版では，ＯＳの核も通常のユーザタスクと同様に，図２のような実行環境を持ち，それらの領域へのアクセスはアドレスレジスタを介して行なわれる．こうしたＯＳの核の実行環境は，ＯＳの核のための TCB により管理される．ＯＳ／ｏ第２版では，このようにＯＳの核もユーザプログラムと同様にタスクとして扱い，統一的な管理を行なう．



UTT (User Trap Table) ：ユーザ拡張部タスクの TCB を管理

図６　カーネルタスクおよびユーザ拡張部タスクの管理

マルチタスクの環境下では，並列に実行中の各タスクが同時にＯＳの核を呼び出すことを考慮しなければならない．これを可能とするため，ＯＳ／ｏ第２版では，ＯＳの核をすべてのユーザタスクから同時に利用できるタスク（これをカーネルタスクと呼ぶ）として扱う．実現方法としては，カーネルタスクの実行環境，およびそれを管理する TCB を，そのとき存在するすべてのユーザタスクの数だけ用意し，それらを図６に示すようにユーザタスクの TCB により管理する．そして，ユーザタスクがＯＳの核を呼び出した場合には，このカーネルタスクを対象として，ディスパッチャがプロセッサを割り当てる．カーネルタスクの実行環境，およびそれを管理する TCB は，ユーザタスクが生成された時点で作られる．

5.1.2　ＯＳの核の呼出し処理

ＯＳの核の呼出しは，MC68000 のトラップ命令(trap #4, trap #5, trap #7)により行なう．ユーザタスクがトラップ命令を実行すると，ディスパッチャは次に示す処理を行なう．

（１）ＯＳの核の実行環境（A1, A2, A3, A4, A5, A6 レジスタ）を回復する．

（２）今まで Running 状態だったユーザタスクの実行環境であるプロセッサのレジスタ値（PC, SR, D0, A1, A2, A7 レジスタ）を，ユーザタスクの TCB に退避する．

（３）カーネルタスクが実行時に使用するスタック領域を設定する．カーネルタスクの実行用スタック領域は，ユーザタスクのスタック領域を引き継いで使用する．

（４）今まで Running 状態だったユーザタスクの TCB を Ready リストの先頭からはずし，代わりにカーネルタスクの TCB をリンクする．カーネルタスクの優先順位は，それを呼び出したユーザタスクの優先順位と同じ値であるため，カーネルタスクの TCB は Ready リストの先頭にリンクされる．

（５）カーネルタスクにプロセッサを割り当てる．処理（１）の段階で，すでにＯＳの核の実行環境となっているため，実際には，各処理ルーチンのエントリへのジャンプ処理となる．

以上がディスパッチャが行なうＯＳの核の呼出し処理である．これに種々のフラグのセット処理を加えて，アセンブラで40ステップで実現されている．ＯＳの核の処理が終了し，ユーザタスクに制御を戻す場合には，上記の処理と逆の処理

を行なう．まず，ＯＳの核の各処理からの戻り値をユーザタスクの TCB の D0 レジスタ退避位置に転送する．次に，カーネルタスクの TCB を Ready リストからはずし，入れ替えにユーザタスクの TCB を Ready リストにリンクする．そして最後に，Ready リストの先頭にリンクされているタスクに対してプロセッサを割り当てる．

5.1.3 割込み処理タスク

ＯＳ／ｏ第２版では，外部装置からの割込みを処理するタスクを割込み処理タスクと呼ぶ．割込み処理タスクは，ＯＳの核とユーザ拡張部に存在することができる．ＯＳの核には，ハードディスク装置，フロッピディスク装置などを対象とした割込み処理タスクが存在する．これらの割込み処理タスクは，ＯＳの立ち上げ時に，ＯＳの核内で生成，実行される．生成された割込み処理タスクは，まず処理対象の外部割込みの確保を行なう．これにより，指定したタスク（自タスク）が外部割込み管理表に登録される．次に，割込み処理タスクは各初期化を行なった後，外部割込みの発生を待つ．

外部割込みが発生すると，ディスパッチャは外部割込み管理表を参照し，その割込みを確保しているタスクを呼び起こす．割込み処理タスクは，もっとも高い優先順位を持つため，すぐにプロセッサが割り当てられ Running 状態となる．

Running 状態となった割込み処理タスクは，Ｉ／Ｏ領域からのデータの読み出し，バッファへの書き込みといった処理を行なう．このバッファへの書き込みの際，同期排他制御手段であるセマフォを利用することにより，バッファからデータを取り出す側のタスク（ユーザタスクから呼び出されたカーネルタスク）と同期処理を行なう．これらの処理を終了すると，再び外部割込み発生待ちとなる．したがって，ループ構造の処理プログラムとなる．

5.1.4 ＯＳの核の実行環境

ＯＳの核は，各ユーザタスクの TCB により管理されるカーネルタスク，割込み処理タスク，そしてアイドリングタスクから構成される．アイドリングタスクは，ＯＳの立ち上げ時に割込み処理タスクと共に生成，実行される．これらＯＳの核を構成する各タスクは，当然ＯＳの核の手続き領域，静的データ領域を共有している．したがって，ＯＳの核は，カーネルタスク，割込み処理タスク，アイドリングタスクをタスクフォースメンバとする一種のタスクフォースとなる．

５．２ ユーザ拡張部の実現

本節では，ユーザ拡張部の登録機構の仕様，およびその実現について述べる．

5.2.1 登録機能

ユーザ拡張部の登録は，__set_ue_task を呼び出すことにより実行される．引数として，登録するプログラムのロードモジュール名，およびユーザ拡張部番号を指定する．ユーザ拡張部番号とは，ユーザタスクがユーザ拡張部を呼び出す際に指定する番号であり，現在０～31が使用できる．ユーザ拡張部の登録を行なうと，登録を行なったタスクが属するタスクフォース内のすべてのタスクで，そのユーザ拡張部を利用できるようになる．また，登録を行なったタスクフォースの子孫にあたるタスクフォースでも利用可能となる．したがって，すべてのユーザタスクから利用できるユーザ拡張部を登録したい場合には，ルートタスクフォースで登録を行なえばよい．

登録されたユーザ拡張部は，それを利用できるユーザタスクがすべて消滅したときに自動的に削除される．また，__reset_ue_task を呼び出すことにより明示的に削除することも可能である．

5.2.2 ユーザ拡張部の利用

ユーザ拡張部の呼出しは，ＯＳの核の場合と同様に，トラップ命令（trap #8 を使用）により行なう．このとき，D0 レジスタの上位ワードにユーザ拡張部番号をセットする．

ユーザ拡張部に登録された処理プログラムは，通常のユーザタスクと同様の手続きでＯＳの核を呼び出すことが可能であり，ファイル，セマフォ，メッセージを使用することもできる．また，ＯＳの核を呼び出す際，そのＳＶＣがＯＳの核にとって，あたかもユーザタスクからのＳＶＣであるかのように見せることができる．この指定は，D0 レジスタの上位ワード（ユーザ拡張部フラグ）の値を ON（１）とすることにより行なう．通常は OFF（０）とする．例えば，ユーザ拡張部フラグを ON として，ファイルのオープンを行なうＳＶＣを発行すると，ファイルはユーザ拡張部にではなく，ユーザタスクに開かれる．もし，ユーザ拡張部でファイルをオープンしたい場合には，ユーザ拡張部フラグを OFF とすればよい．つまり，ユーザ拡張部では，ユーザタスクが行なうＯＳの核へのＳＶＣを，すべて代行することができる．この機構を利用して，ユーザタスクが行なうＯＳの核の呼出しを，すべてユーザ拡張部を介して行なうようにＳＶＣライブラリ関数を実現する．これにより，ユーザタスクのＳＶＣを監視することや，ユーザタスクが呼び出した機能を別の機能に摩り替えるといったことが可能となる．例えば，ユーザタスクがファイルに出力しようとしたデータを，ファイル出力機能をメッセージ通信機能に切り替えることにより，他のタスクフォースにそのデータを受け渡すといったことが実現できる．このように，ユーザ拡張部を利用することにより，標準入出力のリダイレクション機構，またデバッグ対象のＳＶＣを監

視する機能を備えたデバッガなどが実現可能となる．

5.2.3 ユーザ拡張部の実行環境の管理

ＯＳ／ｏ第２版では，ユーザ拡張部に登録された各処理プログラムも，ＯＳの核と同様にタスクとして扱う．これをユーザ拡張部タスクと呼ぶ．ユーザ拡張部タスクの実行環境（プロセッサのレジスタ），およびそれを管理する TCB は，ＯＳの核の場合と同様に，各ユーザタスクごとに存在し，それぞれが UTT (Ueser Trap Table) により管理される（図６参照）．スタック領域については，ユーザタスクのスタック領域を引き継いで使用する．

ユーザ拡張部では，前節で述べたように，通常のユーザタスクと同様にファイル，セマフォ，メッセージを使用することができる．これらの実行環境，そして手続き領域，静的データ領域，ヒープ領域は，同じユーザ拡張部番号を持つユーザ拡張部タスクで共有され，TFCB により管理される．したがって，ユーザ拡張部もまたタスクフォース形態となる．

5.2.4 ユーザ拡張部の呼出し処理

ユーザ拡張部の呼出し処理は，ＯＳの核の場合と同様にディスパッチャが行なう．ユーザタスクがユーザ拡張部を呼び出すためにトラップ命令 (trap #8) を実行すると，ディスパッチャは指定されたユーザ拡張部タスクに対して，プロセッサを割り当てる．ユーザ拡張部タスクが，自タスクの処理を終了してユーザタスクへ制御を戻すための機能が __end_ue_task（表２参照）である．引数としてユーザタスクへの戻り値を指定する．

表２　ユーザ拡張部が使用するＳＶＣライブラリ関数

関数名	機　　能
__end_ue_task	ユーザ拡張部の実行を終了し，ユーザタスクに制御を戻す
__ue_adrs_cnv	ポインタ値の変換を行なう
__lock_inter	外部割込みを確保する
__wait_inter	外部割込みの発生を待つ
__unlock_inter	外部割込みを開放する
__enter_io_seg	I/O 領域の登録を行なう
__lock_task_event	タスクイベントを確保する
__wait_task_event	タスクイベントの発生を待つ

5.2.5 ユーザ拡張部におけるタスクの生成

ユーザ拡張部では，通常のユーザタスクと同様に，タスクフォースメンバを生成することができる．ただし，ユーザタスクの場合と異なり，ユーザ拡張部で生成されたタスクは親子関係を持たない．そして，一旦生成されると，そのタスクを生成したユーザ拡張部が削除されるまで，存在し続けることになる．したがって，ユーザ拡張部におけるこれらのタスクは，ちょうどＯＳの核におけるアイドリングタスク，割込み処理タスクと同様の扱いとなる．

ユーザ拡張部で生成されたタスクは，ＯＳの核内の割込み処理タスクと同様に，外部装置からの割込みを処理することが可能である．処理対象となる割込みを確保する機能，およびそれを開放する機能が __lock_inter，__unlock_inter である．そして，割込みの発生を待つ機能が __wait_inter である．__lock_inter を呼び出すと，指定したタスクが割込み処理タスクとなり，外部割込み管理表に登録される．このとき同時に，タスクの優先順位が最も高い値に設定され，またタスクの実行中の割込みレベルが指定したレベルに設定される．

5.2.6 ユーザ拡張部を利用した日本語入力

ＯＳ／ｏ第２版では，ユーザ拡張部に日本語変換入力機能を実現することにより，プログラミング環境における日本語入力を可能としている．このユーザ拡張部の機能を，日本語変換入力部と呼ぶ．日本語変換入力部では，割込み処理タスクを生成することにより，キーボードからの入力を直接管理している．そして，かな漢字変換機能を持つ一行編集入力機能をユーザタスクに提供する．また，日本語の入力機能の他に，標準入出力のリダイレクション機能，そしてタスクを強制的に終了させるためのキー（アボートキー）の入力を管理する機能を持つ．

６．ＯＳ／ｏ第２版タスク管理の性能評価

マルチタスク機能を持つＯＳの場合，タスクの切り換え処理の実行効率が，ＯＳ全体の性能に大きな影響を与える．ＯＳ／ｏ第２版では，ユーザ拡張部，そしてＯＳの核もタスクとして管理されている．したがって，ＯＳの核の呼出しの場合にも，タスクの切り換え処理が実行される．そこで，これらタスクの切り換え処理に要する命令数，および実行時間を測定することにより，ＯＳ／ｏ第２版におけるタスク管理の性能測定を行なった．測定対象とするシステムは，日立エンジリアニングワークステーション2050/32システム（MC68020：20MHz）上で動作するＯＳ／ｏシステムである．2050/32システムには，多重ＯＳ「江戸」が実現されている[3]．多重ＯＳ「江戸」は，ＯＳの開発，および移行を目的として，複数のＯＳを同時に動作させる機能を持つ．そして，ＯＳに対して MC68000 相当の仮想マシンを提供する．すなわち，ＯＳ／ｏ第２版のプログラム開発環境の役割を持たせた．この多重ＯＳ「江戸」上で動作するＯＳ／ｏ第２版システムを対象として，処理時間の測定を行なった．測定方法としては，ＯＳの核，そしてユーザ拡張部に測定用のルーチンを埋め込み，多重ＯＳ「江戸」が提供するタイマ機能を利用して測定する方法を採った．

まず，ＯＳの核の呼出しに要する時間の測定を行なった．測定対象となる処理内容は，ユーザタスクがトラップ命令を実行してからカーネルタスクの実行が開始されるまでの処理，そしてカーネルタスクの実行が終了してからユーザタスク

の実行が再開されるまでの処理である．したがって，タスクの切り換え処理が２回実行される．測定結果は，表３に示すように約810μsであった．この処理時間には，多重ＯＳによりオーバーヘッドが含まれている．多重ＯＳ「江戸」は，その上で動作するＯＳをユーザモードで実行させ，ＯＳが特権命令を実行すると，それをソフトウェアでエミュレートしている．これによるオーバーヘッドが約670μs(特権命令を６命令実行）であった．したがって，ＯＳ／ｏが行なう処理時間は実質約140μsとなり，約80％が多重ＯＳによるオーバーヘッドとなる．

この結果からも明らかなように，ＯＳ／ｏ第２版は，ほぼ満足できる性能を得た．もちろん，多重ＯＳを用いないマルチプロセッサシステムは，ＯＳ／ｏ第２版の性能をより引き出すことができる．以下に述べる性能評価は，多重ＯＳによるオーバーヘッドを除いた結果に対するものである．

ユーザタスクからカーネルタスクへの切り換え処理は，第5.1.2節で述べたように40命令で実行される．それに対して，カーネルタスクからユーザタスクへの切り換え処理は79命令である．この原因は，前者がアセンブラで記述されているのに対し，後者はその一部が言語Ｃで記述されているためである．この部分の最適化が，今後の課題となるであろう．

次に，ユーザ拡張部の呼出し処理に要する時間の測定を行なった．測定対象となる処理内容は，ＯＳの核の呼出し処理の場合と同様に，ユーザタスクからユーザ拡張部タスクへの切り換え，そしてユーザ拡張部タスクからユーザタスクへの切り換え処理である．ユーザ拡張部タスクが実行を終了し，ユーザタスクに制御を戻すためには，ＯＳの核に対してＳＶＣを発行する必要がある．このため，ＯＳの核の呼出しに要する時間と比較して，多少のオーバーヘッドが予想できる．

測定結果は表３に示すように約1620μsであった．この値から，多重ＯＳによるオーバーヘッドを除くと，ＯＳ／ｏの処理時間は約410μsとなる．これを，ＯＳの核の呼出し処理の結果と比較すると，処理時間が約270μs多くかかっていることになる．したがって，ユーザ拡張部で拡張機能を実現した場合には，ＯＳの核で実現した場合と比較すると，その呼出しの際に，約270μsのオーバーヘッドを伴うことになる．しかし，ユーザ拡張部の動的登録機構により得られる利点を考えれば，このオーバーヘッドは充分小さな値と言えるであろう．

これらの結果を踏まえ，ユーザタスクに提供するＳＶＣライブラリ関数は，すべてユーザ拡張部（ユーザ拡張部番号０）を通して，ＯＳの核を呼び出す仕様で実現した．これにより，第5.2.2節でも述べたように，ユーザタスクのＳＶＣを監視する機能などを，ユーザ拡張部を利用して容易に実現可能となる．この仕様により，ユーザタスクにとってのＳＶＣの実行時間は，550μs（ＯＳの核の呼出し処理時間＋ユーザ拡張部の呼出し処理時間）以上となる．セマフォに対するＰＶ命令の実行時間を測定したところ，表３に示すような結果が得られた．これらの測定値から多重ＯＳによるオーバーヘッドを除くと，それぞれ約720μs，約690μsの処理時間となる．また，ユーザ拡張部タスクがＯＳの核を直接呼び出してＰＶ命令を実行した場合の処理時間を測定した結果，約310μs，約280μsであった．これらの値は，充分実用に耐えうるものと言えるであろう．

表３　測定結果

処理内容	処理時間	
	多重OSの処理を含む	多重OSの処理を除く
OSの核の呼出し処理	810μs	140μs
ユーザ拡張部の呼出し処理	1620μs	410μs
ユーザ拡張部を通した場合		
Ｐ命令の実行	2560μs	720μs
Ｖ命令の実行	2530μs	690μs
OSの核を直接呼び出した場合		
Ｐ命令の実行	950μs	310μs
Ｖ命令の実行	920μs	280μs

７．おわりに

ＯＳ／ｏ第２版のプログラムの規模は，現在，言語Ｃで記述した部分が約25000行，アセンブラで記述した部分が約500行になっている．そのうち，タスク管理部は言語Ｃが約5000行，アセンブラが約500行である．ＯＳ／ｏ第２版は，現在研究室においてプログラム開発システムとして実際に稼働している．本システムの実現により以下の成果が得られた．

（１）一部の実行環境を共有したタスクフォースと呼ばれるタスクの形態を実現した．これにより，高速でかつ柔軟なタスク間通信が可能となった．

（２）ＯＳの機能拡張，およびデバイスハンドラの登録を動的に行なう機構を，ユーザ拡張部という形で実現した．

（３）これらの成果により，種々のデバイスを用いたマンマシンインタフェースの研究，そして並列処理記述言語の研究の基盤となる環境を提供できた．

参考文献

[1] JIS ハンドブック 情報処理-1987，日本規格協会

[2] 並木美太郎　他，“言語Cコンパイラcatの方式設計”，情報処理学会ソフトウェア工学研究会資料48-2，8頁，1986.6

[3] 岡野裕之　他，“多重ＯＳ「江戸」とそのウィンドウ・システムの開発”，情報処理学会オペレーティング・システム研究会，39-4，1988.6